

The OWL API: An Introduction



Sean Bechhofer
University of Manchester
sean.bechhofer@manchester.ac.uk

Overview

- Motivation
 - Why?
- Details
 - What? Where?
- Samples/Examples
 - How?
- Wrap Up
 - What next?

Assumptions

*"The least questioned assumptions
are often the most questionable"*
Paul Broca

- Familiarity with Java
- (Some) Familiarity with Semantic Web Technologies:
 - RDF
 - RDF Schema
 - OWL
- Being at least aware of the existence of:
 - Description Logics

COMP60462 3

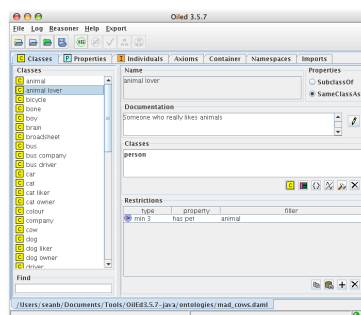
Points from History

- Influence of frame based modelling approaches
 - Classes, slots, fillers
- Influence of logical foundations
 - Well-formed semantics
 - Inference
- Influence of Web Languages
 - RDF, RDF(S)

COMP60462 4

History

- **OilEd**
 - One of the first ontology editors to employ description logic reasoning
 - Targeted at OIL (then DAML+OIL)
 - ~10,000 registrations for download
 - Used for teaching in both academia and industry
- Data model independent of the underlying concrete syntax
 - But without decent abstractions



COMP60462 5

History/Provenance

- **WonderWeb**
 - <http://wonderweb.semanticweb.org/>
 - An EU STREP targeted at producing *Ontology Infrastructure for the Semantic Web*
 - Produced the initial implementations of the API
- **CO-ODE**
 - <http://www.co-ode.org/>
 - UK JISC funded project to provide support for communities developing OWL
 - Protégé tool development and OWL support
 - Supporting current development.

COMP60462 6

OWL

OWL allows us to describe a domain in terms of:

- **Individuals**
 - Particular objects in our domain
- **Classes**
 - Collections of objects (usually sharing some common characteristics)
- **Properties**
 - Binary relationships between individuals.
- Plus a collection of **axioms** describing how these classes, individuals, properties etc. should be interpreted

Note: This talk will not be discussing whether this is the “right” way to do things....

COMP60462 7

OWL

- OWL has a number of operators that allow us to describe the classes and the characteristics that they have
- Boolean operators
 - and, or, not
- Quantification over properties/relationships
 - universal, existential.
- A clear and unambiguous semantics for the operators and composite class expressions

COMP60462 8

OWL and RDF

- The normative presentation for OWL was based on RDF
- XML-RDF is one of a number of possible syntactic presentations of OWL.
- Working solely at the RDF triple level can introduce both mechanical and conceptual difficulties.
 - Round tripping. The mapping rules between OWL abstract syntax and triples are not 1:1.
 - Containment. Assertions that statements belong to a particular ontology.
- Alternative syntaxes can be useful
 - OIL's original text format was popular with users – you *can* write XML-RDF in emacs, but I wouldn't recommend it.....

COMP60462 9

OWL and RDF

- Sometimes an uneasy relationship between the two
- Syntactic Layering
 - Additional baggage/hoops to jump through
- Semantic Layering
 - Complicated

OWL != RDF

- One of our key motivators was to provide infrastructure that helped insulate applications from these issues

COMP60462 10

Why build an OWL API?

- The use of a higher level data model can help to
 - insulate us from the vagaries of **concrete syntax**.
 - make it clear what is happening in terms of **functionality**.
 - increase the likelihood of **interoperating** applications.

COMP60462 11

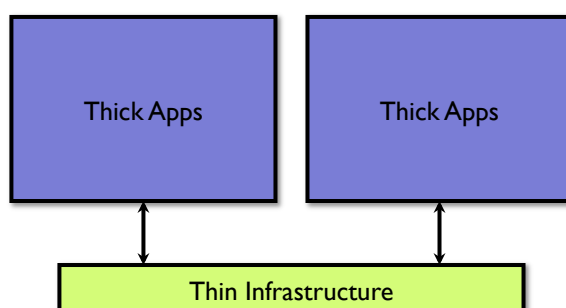
Experiences

- An early example application built using the API was a web service that performed validation and translation to alternative syntaxes.
 - See *OWL DL: Trees or Triples*, WWW2004
- Ontology level objects made it easy to write code spotting “internal errors”

COMP60462 12

How thick is **your** infrastructure?

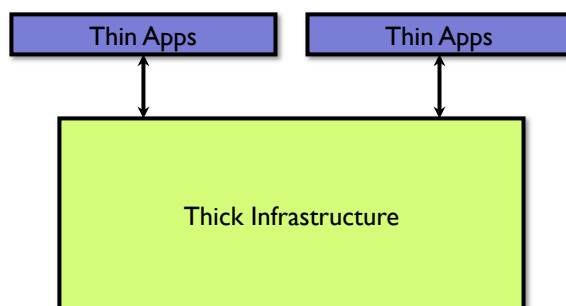
- A lightweight infrastructure (e.g. RDF) means that clients/apps have to do more. And *may* do it differently.
- Metadata can end up being locked away within the applications where others can't get at it. Is that **sharing**? Are you exposing the **semantics**?



COMP60462 13

How thick is **your** infrastructure?

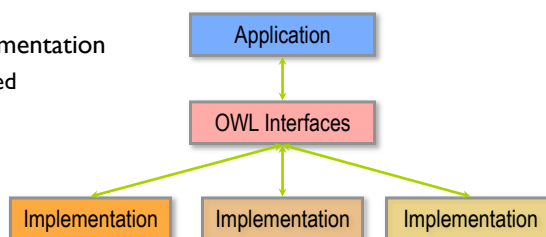
- Sharing is about interoperations. Ensuring that when you look at or process my data, you do it in a consistent way.
- “Thick” infrastructure can help interoperability. Clients don't have to guess how to interpret things.
 - But can be harder to build



COMP60462 14

Assumptions

- Primarily targeted at **OWL-DL**
 - This does not mean that we **cannot** handle OWL-Full ontologies, but a number of design decisions reflect this assumption.
- Java based
 - Interfaces
 - Java **reference** implementation
 - **Main memory** based



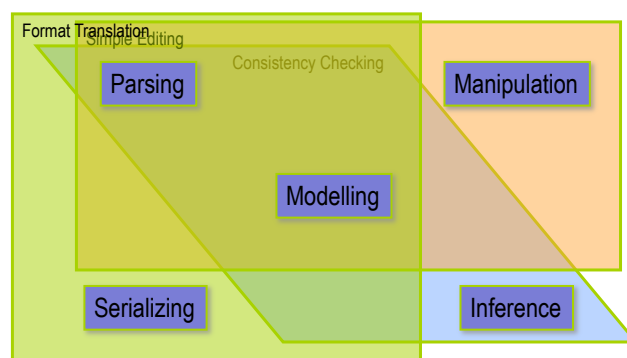
COMP60462 15

What is an “OWL Implementation”?

- **Modelling**
 - Provide data structures that represent OWL ontologies/documents.
- **Parsing**
 - Taking some syntactic presentation, e.g. OWL-RDF and converting it to some [useful] internal data structure.
- **Serializing**
 - Producing a syntactic presentation, e.g. OWL-XML from a local data structure.
- **Manipulation/Change**
 - Being able to manipulate the underlying objects.
- **Inference**
 - Providing a representation that implements/understands the formal semantics of the language.

COMP60462 16

Implementation Aspects



COMP60462 17

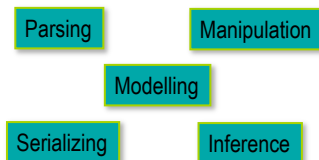
OWL Structural Specification

- Provides a definition of the language in terms of the constructs and assertions allowed.
- **Semantics** are then defined in terms of this abstract syntax.
- Our OWL API data model is based largely on this abstract presentation.
 - Conceptually cleaner.
 - Syntax doesn't get in the way

COMP60462 18

Considerations

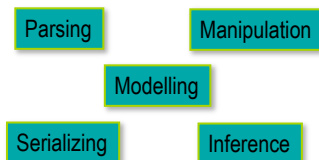
- Clear identification of functionalities and a separation of concerns
- Representation
 - Syntax vs. Data Model
 - Interface vs. Implementation
 - Locality of Information
- Parsing/Serialization
 - Insulation from underlying concrete presentations
 - Insulation from triples



COMP60462 19

Considerations

- Manipulation/Change
 - Granularity
 - Dependency
 - User Intention
 - Strategies
- Inference
 - Separation of explicit assertions from inferred consequences
 - External reasoning implementations



COMP60462 20

Programming to the OWL API: The API



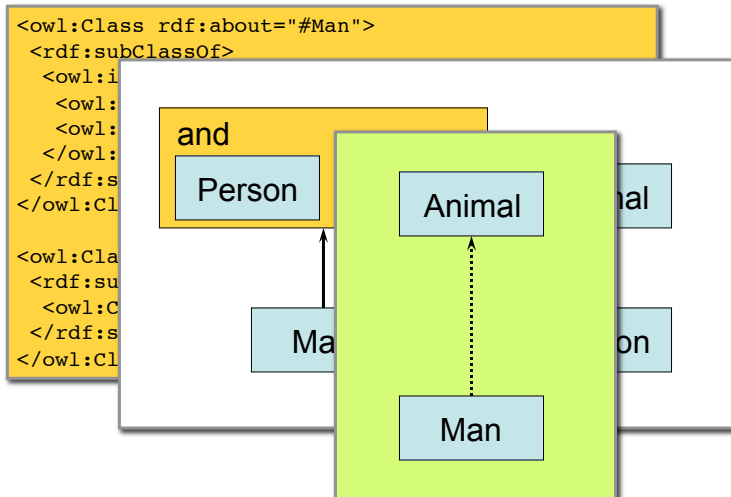
Sean Bechhofer
University of Manchester
sean.bechhofer@manchester.ac.uk

What is an Ontology?

- A particular syntactic presentation
- The facts represented by that syntactic presentation
- The information that can be inferred from those facts
 - E.g. consequences of the assertions plus the underlying semantics of the language.
- The difference between these becomes important with languages like OWL.
- What answers will I expect when interrogating the structure?

COMP60462 22

What is an Ontology?



COMP60462 23

Overall Philosophy

- The OWL API is targeted primarily at representing OWL-DL
- An Ontology is represented as a collection of axioms that assert information about the classes, properties and individuals that are in the ontology
 - Provides a uniform view on the ontology
- *When is a class or property “in” an Ontology?*
 - This isn't explicitly captured in the OWL specs
 - Somewhat application dependent

COMP60462 24

Basic Data Structures

- At its heart, the OWL API provides data structures representing OWL ontologies
- Plus classes to help
 - Create;
 - Manipulate;
 - Parse;
 - Render; and
 - Reason about those structures
- The basic data structure represents the objects in the ontology and corresponds roughly to the abstract syntax of OWL.

COMP60462 25

OWLOntology

- An OWLOntology represents an ontology
- It consists of a collection of OWLAxioms
- Note that the OWLOntology object doesn't explicitly contain classes, properties etc.
 - The classes "in" the ontology are those that are referenced by the axioms that are in the ontology
 - We believe this provides a *clean* and *consistent* story as regards the objects that make up the ontology
- The OWLOntology provides an explicit context within which we can assert information about classes and properties.
 - Locality of information

COMP60462 26

Names and URIs

- Ontologies in OWL are **named** using URIs
- Ontologies can also be **retrieved** from URLs.
- This can cause some confusion!
 - There's no explicit requirement that the location you retrieve the ontology from is the same as the name that it's given.
- The situation isn't made any easier by OWL's import mechanism.

COMP60462 27

OWLEntity

- OWLEntity is the fundamental building block of the ontology
 - Classes
 - Properties
 - Individuals
 - Datatypes
- Named using URIs
 - But be aware of **punning**

COMP60462 28

OWLClass

- Represents an OWL Class.
- The Class itself is a relatively lightweight object
 - A Class doesn't hold information about definitions that may apply to it.
- Axioms relating to the class are held by an OWLOntology object
 - E.g. a superclass axiom must be stated within the context of an OWLOntology
 - Thus alternative characterisations/perspectives can be asserted and represented for the same class.
- Axioms do not relate only to a class, but to a class within the context of an ontology

COMP60462 29

OWLClass

- Methods are available on OWLClass that give access to the information within a particular ontology

```
java.util.Set<OWLDescription> getDisjointClasses(OWLOntology ontology)
java.util.Set<OWLDescription> getEquivalentClasses(OWLOntology ontology)
```

- But these are simply *convenience* methods.

COMP60462 30

OWLProperty

- OWL makes a distinction between
 - Object Properties: those that relate two individuals
 - E.g. hasBrother
 - Data Properties: those that relate an individual to a concrete data value
 - E.g. hasName
- There is a strict separation between the two and two explicit classes representing them
 - OWLObjectProperty
 - OWLDataProperty

COMP60462 31

OWLProperty

- Properties can have associated domains and ranges
- There is also a property hierarchy
 - Super properties
 - Property equivalences
 - Disjoint Properties (OWL2)
- In OWL2, we also have the option of using property expressions (e.g. property chains).
- Again, as with classes, the property objects are lightweight and all assertions about properties are made in the context of an Ontology.
 - E.g functional properties

COMP60462 32

OWLObjectProperty

- Represents an Object Property that can be used to relate two individuals
- Object properties can have additional characteristics
 - Transitivity
 - Inverses

COMP60462 33

OWLObjectProperty

- Represents a Data Property that can be used to relate an individual to some concrete data value
- Data properties can also have additional characteristics
 - Functional

COMP60462 34

OWLDescription

- OWLDescription represents an OWL class expression
- Atomic classes
- Boolean expressions
 - Intersection (and)
 - Union (or)
 - Complement (not)
- Restrictions
 - Explicit quantification (some, only)
 - Cardinality restrictions (min, max)

COMP60462 35

OWLAxiom

- An ontology contains a collection of OWLAxioms
- Each axiom represents some fact that is explicitly asserted in the ontology
- There are a number of different kinds of axiom
 - Annotation Axioms
 - Declaration Axioms
 - Import Axioms
 - Logical Axioms

COMP60462 36

Annotation Axioms

- An `OWLAnnotationAxiom` is used to associate arbitrary pieces of information with an object in the ontology
 - Labels or natural language strings
 - Dublin core style metadata, e.g. author or creator information
- Annotation Axioms have no logical significance
 - They do not affect the underlying semantics of the ontology

COMP60462 37

Declaration Axioms

- An `OWLDeclarationAxiom` simply declares or introduces an entity into an ontology
- Declaration axioms have no logical significance
- Such axioms can be useful for integrity checks, e.g. tools may require that every entity used in the ontology is explicitly declared
 - This can help detect issues or problems with mis-spelling

COMP60462 38

Import Axioms

- An `OWLImportsDeclaration` annotates an ontology in order to describe the ontologies it imports.
 - In the OWL specification, `owl:import` is a kind of `owl:OntologyProperty` which is treated in a special way.
- As discussed earlier, we need to be careful about the way in which we treat ontology import structures.

COMP60462 39

Logical Axioms

- The subclasses of `OWLLogicalAxiom` represent the logical assertions contained in the ontology
 - Supers (of classes and properties)
 - Equivalences (of classes and properties)
 - Property Characteristics
 - Functionality, transitivity etc.
 - Facts about particular individuals
 - Types
 - Relationships
 - Values

COMP60462 40

Equality

- Equality on objects in the API is defined structurally.
- In general, two objects are equal if they have the same structure
- E.g. for boolean class descriptions, the operands are compared as a Set.
 - If the two sets of operands are equal, the descriptions are equal.
- Note that this is syntactic. Logical equivalences that may follow from axioms or assertions in the ontology do not impact on the equality of the objects.

COMP60462 41

Visitors

- The API makes extensive use of the **Visitor** Pattern
 - Represents an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates
- Design Patterns (Gamm et al)
- The use of the Visitor pattern allows us to define many different operations over the data structure without “polluting” it with application specifics.
 - Visitor is good when the data structure is static, but costly if the structure changes
 - As our data structures are based on the language, the structure is likely to remain static here.

COMP60462 42

Visitors

- Visitors are primarily used when rendering or serializing ontologies into different concrete formats.
- There are Visitors for each of the main classes
 - OWLEntityVisitor
 - OWLAxiomVisitor
 - OWLDescriptionVisitor
- Visitors are also used to enact changes to the Ontology
 - OWLOntologyChangeVisitor

COMP60462 43

The times they are a changing...

- The data structures provide read-only access to the information held in the ontology
 - Changes to the ontology are effected through the use of the **Command** Pattern.
 - *Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations*
- Design Patterns (Gamma et al)
- Objects explicitly represent the changes
 - The change objects are passed to a visitor which then enacts the changes

COMP60462 44

Changes as Objects

- Undo operations
- Metadata representing the **provenance** of the change
- Explicit **dependencies** between changes
- Representing change sets
 - History
 - Checkpoints
- Use of the **Strategy** Pattern to enact changes
 - *Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.*

Design Patterns (Gamma et al)

COMP60462 45

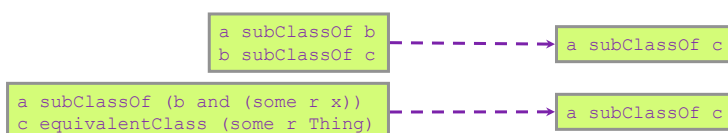
Changes

- The API takes an “axiom-centric” view
- There are a limited number of change objects
 - Add an Axiom
 - Remove an Axiom
 - Set the Ontology URI
- Trade off between simplicity and power
 - Change from original API, which had a number of different change objects encapsulating different changes.
 - Change object describes what happened, e.g. add/remove
 - Wrapped axiom describes the change
- Changes then enacted by a Change Visitor.

COMP60462 46

Inference

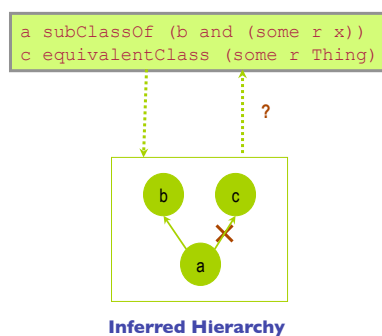
- OWL's semantics allows the possibility to perform inference or reasoning over an ontology
- A reasoner may be able to determine additional facts that follow from the information in the ontology
- How best do we expose this information?
 - Add it into the structure?
 - Provide alternative interfaces?



COMP60462 47

Inference

- This is particularly important when users get involved
 - What do we do when the user tries to **remove** some inferred information?
- **Separating** the inferred information from the asserted information doesn't necessarily answer that question
 - But does mean that we have **enough** to be able to make a choice.
- Offering separate inference interfaces also allows applications to be **clear** about the services they may (or may not) supply.



COMP60462 48

Inference

- The API includes a collection of interfaces that provide access to inference services operating over the model
- `OWLClassReasoner`
 - Hierarchy information, class satisfiability
- `OWLConsistencyChecker`
 - Ontology consistency
- `OWLIndividualReasoner`
 - Reasoning about individuals
- `OWLPropertyReasoner`
 - Reasoning about properties

COMP60462 49

Reasoner Implementations

- Generic DIG Reasoners
 - Via DIG I.I
- Pellet
 - Pure Java implementation
 - Implements OWL API reasoner interfaces
- FaCT++
 - C++ Implementation
 - Java wrapper
 - OWLAPI wrapper implementing OWL API interfaces
- Debuggers etc. then sit on whichever reasoner you have.
- Syntactic/Told Reasoners

COMP60462 50

Putting things together

- So far, what we've seen are a collection of interfaces providing access to the various aspects of functionality that we can associate with OWL ontologies
- How do we bring this all together in order to actually *use* the code and make use of a particular implementation?

COMP60462 51

Managing Ontologies

- The model data structures provide representations of the basic building blocks.
- Management and creation of ontologies is controlled by an OWLOntologyManager
- The Manager is responsible for keeping track of the ontologies and concrete formats for storage of the ontologies.
- Handles Ontology changes

COMP60462 52

One Ring to Bind Them

- The apibinding module provides a utility class OWLManager that brings together common parsers and renderers along with the in-memory implementation
- We can use this as a point of entry into the implementation.
 - Provides basic infrastructure that takes care of standard
 - If we're doing "standard" things, then the basic OWLManager setup is enough.
 - We can extend this, by, for example adding new concrete syntaxes (along with parsers and renderers that will handle those formats).

COMP60462 53

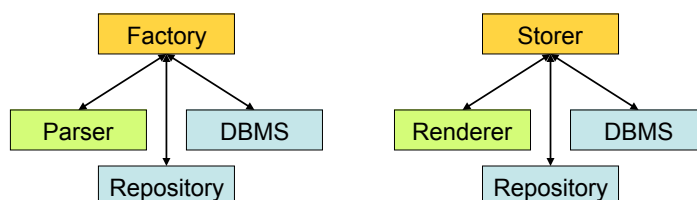
URI Mapping

- A URIMapper allows us to maintain a separation between logical URIs and physical URIs
- Useful for
 - Offline working
 - Caching ontologies
 - Solving the name/location problems identified with imports

COMP60462 54

Factory and Storer

- OWLOntologyFactory and OWLOntologyStorer are used for loading and saving ontologies
- They provide an extra level of abstraction above parsers and serialisers/renderers
 - Allows the potential for retrieving ontologies from data sources other than files/streams, e.g. DBMS/repository/RMI etc.



COMP60462 55

Parsing and Rendering

- Parsers and Renderers allow creation of ontologies from concrete serialisations and storage of ontologies using those serialisations.
- A Parser returns information about the format that the ontology was parsed in.

COMP60462 56

Loading an Ontology

- Map the logical URI to a physical URI
- Query available Factories for one that can handle the URI
- For the basic factories in our implementation, the selected Factory will then use a parser to parse the given URI
 - Selection of parsers could be via MIME-types or extension mappings
 - Currently simple approach
 - Could do something other than parsing....

COMP60462 57

Ontology Formats

- The `OWLOntologyFormat` class represents a format used for concrete serialisation
 - E.g. OWL RDF/XML
- The format may also contain information about the particular serialisation
 - E.g. namespace declarations
 - Ordering
 - Structural information
 - Helps in addressing problems with round-tripping
- If an ontology was parsed, the `Manager` maintains information about the original format of the ontology

COMP60462 58

Manager

- The default Manager provides support for a number of basic formats
- Parsing:
 - RDF/XML; OWL XML; KRSS; OBO; Functional Syntax
 - ParserFactoryRegistry knows about available parsers.
- Rendering
 - RDF/XML; OWL XML; Functional Syntax
 - Most renderers effectively involve writing a Visitor that walks the data structure
 - See later example
- You *could* do it all yourself with bespoke parser factories etc....

COMP60462 59

Recap

- Basic interfaces providing data structures
 - Visitors
- Manipulation via Change objects
- Inference via parallel interfaces
- I/O through Factory and Storer classes, wrapping Parsers and Renderers
- Manager object pulling together standard components.

COMP60462 60

Programming to the OWL API: Examples



Sean Bechhofer
University of Manchester
sean.bechhofer@manchester.ac.uk

Nuts and Bolts

- OWL API code is available from sourceforge:
<http://sourceforge.net/projects/owlapi>
- Latest versions are in the SVN repository.

Modules

- The distribution is split into a number of different modules
 - Each module has some identified functionality.
 - There may be dependencies between modules.
- The distribution uses Apache's maven to control the build process

COMP60462 63

Modules

- api
 - Core interfaces covering ontology data structures, parsing and rendering, inference etc.
- impl
 - Basic “reference” implementation
- xxxparser
 - Parser for concrete syntax xxx (e.g. rdfxml, krss, owlxml, obo etc)
- xxxrenderer
 - Renderer for concrete syntax xxx (e.g. rdfxml, krss, owlxml etc)
- debugging
 - Inference debugger

COMP60462 64

Modules

- `digl_l`
 - Classes supporting interaction with DIG reasoners
- `rdfapi`
 - RDF parser (from KAON)
- `apibinding`
 - Provides a point of convenience for creating an `OWLOntologyManager` with commonly required features (e.g. `rdfoxmlparser`).

COMP60462 65

Examples

- The following examples serve to illustrate various aspects of the API and how we might use it.
 1. Producing a basic hierarchy
 2. Rendering Alternative Concrete Syntaxes
 3. Adding Closure Axioms
 4. Black Box Debugging
- We won't go into *all* the details of how these are done
 - Source for the examples will be available on line.

COMP60462 66

I. Hierarchy Example

- Our first example will read an ontology, use a reasoner to calculate the inferred subsumption hierarchy, and then display this hierarchy.

COMP60462 67

Details

1. Creating/reading ontologies
 2. Basic reasoner interfaces
 3. Instantiating a reasoner
 4. Printing out the results
- Uses the basic **apibinding**
 - Standard parsers.
 - Interrogation of the data structures
 - Visitor for simple display of entities.

COMP60462 68

2. Concrete Syntax

- There are a variety of concrete syntaxes available for OWL
 - XML/RDF, XML Schema, Concrete Abstract Syntax
- Here we'll show how we can add a serialisers for a new syntax.

COMP60462 69

Details

1. Defining renderer for objects
 2. Defining renderer for ontology
 3. Plugging it into the manager infrastructure
- Definition of some Visitors
 - Format object and associated storage helpers
 - Registration with the manager.

COMP60462 70

3. Closure Axioms

- OWL allows us to provide definitions of classes using composite expressions to assert necessary and sufficient conditions.
- However, it's not always easy to capture exactly what we want.
- Closure axioms are an example where additional tool support can help to achieve the “right” model.

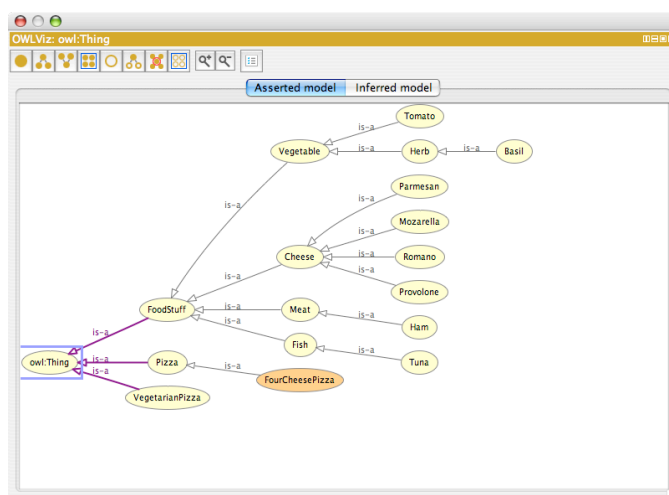
COMP60462 71

Pizzas

- Tomato is a Vegetable
- Basil is a Herb (which is a Vegetable)
- Cheese, Vegetables and Herbs are disjoint from Meat and Fish
- Mozzarella, Provolone, Romano and Parmesan are all kinds of Cheese.
- A FourCheesePizza has Tomato, Mozzarella, Provolone, Romano and Parmesan and Basil.
- A VegetarianPizza is one that has no Meat or Fish toppings
- Is a FourCheesePizza a VegetarianPizza?

COMP60462 72

An example Model



COMP60462 73

Pizzas

- The problem is that, although we have given conditions that describe what toppings a **FourCheesePizza** must have, the possible world semantics that OWL has means that we can't preclude situations where a **FourCheesePizza** has other things on it.

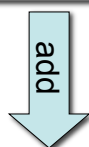


COMP60462 74

Closure Axioms

- A standard pattern to deal with this situation is to add a “Closure axiom” that states that, in addition to the existential restrictions, there is a universal restriction stating that the only possible fillers are those named.

`X subClass ((some p A) and (some p B) and (some p C))`



`X subClass (all p (A or B or C))`

COMP60462 75

Adding Closure axioms

1. Find the axioms that define C
 2. Filter subclass axioms
 3. Pull out any existential restrictions
 4. Create a new axiom with a universal restriction and a filler formed by the union of the definitions
 5. Add the axiom to the ontology
- We can do this via a combination of
 - OWLOntology methods
 - Specialised visitor to collect information about axioms
 - Change objects

Working with
axiom objects

COMP60462 76

4. Black Box Debugging

- Constructing OWL ontologies can be a difficult task.
- Unexpected inferences may be derived
- Debugging can help to track down the reasons for any subsumptions or inconsistencies.
- The API includes interfaces to support debugging of ontologies
 - Based on work from Maryland's Mindswap Lab.
- We can use this to provide an explanation service

COMP60462 77

Details

1. Read ontology
2. Determine Unsatisfiable Classes
3. Provide explanation for the unsatisfiability
 - Minimal sets of axioms causing unsatisfiability
 - Possibly multiple sets
4. Report sets of axioms
 - Rendering again

COMP60462 78

Protégé and the API

- Protégé is an ontology editor.
 - One of the most used ontology editing tools (for OWL at least)
- Originally built on top of a frame-based model
- OWL support added through a layered approach.
 - Somewhat problematic
- A reimplementation (Protégé 4) is targeted specifically at OWL and is built on top of the OWL API

COMP60462 79

Protégé and the API

- Load/Store Abstractions
 - Allows much cleaner client code.
 - In the past, the tool had to worry about a lot of the details regarding input/output formats
 - E.g. what do you do when the user hits “Save”.
- Change
 - Recording history/checkpoints
 - Tracking ontology changes
 - Undo!
 - Event notification
- Can still support a frame-based presentation on top of the axiom-centric model.

COMP60462 80

SWOOP and the API

- SWOOP is an OWL editor, originally from the University of Maryland.
- It's a "hypermedia inspired" lightweight editor for OWL ontologies
 - Web browser look & feel
 - Inline Editing
 - Integration with a reasoner
- SWOOP uses the OWL API to represent OWL Models.

COMP60462 81

SWOOP and the API

"...Swoop uses the OWL API to model ontologies and their associated entities, benefiting from its extensive and clean support for changes. The OWL API separates the representation of changes from the application of changes. Each possible change type has a corresponding Java class in the API, which is subsequently applied to the ontology (essentially, the Command design pattern). These classes allow for the rich representation of changes, including *metadata* about the changes."

COMP60462 82

Programming to the OWL API: Wrap Up



Sean Bechhofer
University of Manchester
sean.bechhofer@manchester.ac.uk

The OWL API

- OWL is a language for representing ontologies.
- Although it has a relationship with other representations like RDF, it has particular characteristics
- An API allows tools to work at an appropriate level of abstraction
 - Working with the language objects
 - Hiding issues of concrete syntax and representation
- Provides clear identification of functionalities
 - Particularly important when we have inference

COMP60462 84

Highlights

- Basic data structures representing OWL Ontologies
 - Influenced by the logical perspective
 - Targeted at a particular level of expressivity
 - OWL DL/OWL I.I
- Parsing & Rendering
 - Support for standard formats (e.g.XML/RDF)
- Manipulation
 - Through explicit change objects
- Inference
 - Explicit Reasoning interfaces
 - Communication with reasoners through “standard” protocols

COMP60462 85

Where's it used?

- | | |
|--|---|
| • Pellet <ul style="list-style-type: none"> – OWL reasoner | • OntoTrack <ul style="list-style-type: none"> – OWL Editor |
| • SWOOP <ul style="list-style-type: none"> – OWL editor | • DIP Reasoner |
| • Protégé 4 <ul style="list-style-type: none"> – OWL editor | • BT <ul style="list-style-type: none"> – SNOMED-CT support |
| • ComparaGrid | • BioPAX <ul style="list-style-type: none"> – Lisp bindings (!!) |
| • CLEF | |

COMP60462 86

Caveats

- Primarily designed to support manipulation of T-Box/
schema level ontologies
 - Large amounts of instance data may cause problems.
- Designed to support OWL (not RDF)
- This isn't industrial production level quality code
 - It's not bad though :-)
- We can't promise to answer all your questions
- We can't promise to fix all your bugs
- But we'll try.....

COMP60462 87

The Future

- The API continues to be developed, primarily to support the development of Protégé and OWL support within Protégé.
- Additional Components to support UIs and general manipulation
 - Migrating Protégé code into the code base
- Handling Change History
- Alternative implementations
 - DB backed storage?
- Query Interfaces
 - Conjunctive Query Style?

COMP60462 88

Other, Related Work

- Jena
 - Provides OWL Ontology interfaces, layered on the RDF structures of Jena
- Protégé API
 - Protégé 3 provided OWL API layered on Protégé model
 - Mixture of frames, RDF and OWL
 - Evolution to support a UI
- KAON2
 - Support for OWL
 - Not open source

COMP60462 89

References

- *Cooking the Semantic Web with the OWL API*, ISWC2003
- *Parsing OWL DL: Trees or Triples?*, WWW2004
- *Patching Syntax in OWL Ontologies*, ISWC 2004
- *The Manchester OWL Syntax*, OWLEd 2006
- *Igniting the OWL 1.1 Touch Paper: The OWL API*, OWLEd 2007
- *The OWL API: A Java API for Working with OWL 2 Ontologies*, OWLEd 2009

COMP60462 90

Thanks!

- Thank you for listening
- Thanks also to everybody who has contributed to the API design and codebase
 - Matthew Horridge
 - Phil Lord
 - Angus Roberts
 - Raphael Volz
 - Olaf Noppens
 - Evren Sirin
 - Any others I forgot to mention (sorry!)...

COMP60462 91

More Information

<http://sourceforge.net/projects/owlapi>
<http://owlapi.sourceforge.net/>
<http://www.co-ode.org/>
<http://owl.cs.manchester.ac.uk/2009/07/sssw/>

COMP60462 92